Functions and Pseudocode :

void insertfheap(record *temp)		
Description	Inserts records into the fibonacci heap	
Parameters	temp	Pointer of the new node to be inserted
Return Value	No return value	

Pseudocode:

```
insertfheap(){
        if(empty)
                Add new record
        Else
                Add to the previous record
}
```

FiboHeap

The methods to implement the Fibonacci heap are defined in this class. All operations in this class are performed on instances of FiboHeapNode class.

The member functions of this class are:

- **FiboHeapNode insert(FiboHeapNode n):** Used to insert a new node into the Fibonacci heap.
- **void increaseKey(FiboHeapNode n, int k):** Used to increase the value of n to k which is passed as an argument to the function.
- **void cascadingCut(FiboHeapNode t):** Checks the childCut value and performs necessary changes and removes the

node and adds it to the root level if necessary.

- **void cut(FiboHeapNode a, FiboHeapNode b):** Removes the node from the doubly linked list.
- **FiboHeapNode removeMax():** Removes the node with the maximum frequency from the Fibonacci heap.
- **setNewMaxPointer():** Updates the pointer to the node with maximum value in the Fibonacci heap.
- **void pairWiseCombine():** Combines two nodes of same degree into a single heap. pairWiseCombine() is performed every time an increaseKey() operation is performed.

record* meld(record* first,record* second)		
Description	Combines two records to form a new record with increased degree	
Parameters	first	Always input highest value pointer as first
	second	Always input lowest value pointer as second
Return Value	Returns “first” pointer	

void AddTotable(record temp,unordered_map<int,record> &ordertable)		
Description	add to table based on degrees and perform melding	
Parameters	temp	Pointer to add into the degree table
	ordertable	Pointer to the hashmap which hold degrees and their record pointer
Return Value	No return value	

The member functions of this class are:

- **int getKey():** It returns the value of the Fibonacci Node.
- **String getTag():** It returns the Hashtag.
- **void link(FiboHeapNode a,FiboHeapNode b):** Takes care of all the links in the doubly linked list whenever a cut is performed.

FiboHeapNode

The class FiboHeapNode mainly describes the structure of the node that is used in the implementation of the Fibonacci Heap.

The variables each represent the attributes of a node. They are:

- **element:** element stores the hashtag.
- **key:** key stores the Fibonacci node number.
- **degree:** Stores the number of nodes that are present in the next level.
- **childCut:** Specifies whether the node has already lost a child or not. It is of Boolean type.

This class has other variables such as parent, child, right and left which are used for the circular doubly linked list that is created at each level.

record cutnode(record* temp)		
Description	remove the record given and insert as root	
Parameters	temp	Pointer to which record to be removed and inserted as root
Return Value	Return pointer to the parent of the record before removing	

void increment(record temp,int value)		
Description	increase value and perform childcut	
Parameters	temp	Pointer to the node to be incremented (you get that from hashmap)
	value	How much value you need to increase
Return Value	No return value	

Pseudocode:

```
increment(record *temp,int value){
        if(root node is incremented)
                Change value and update max if needed
        Else
                while(temp->parent!=NULL){
                        temp=cutnode(temp)
                        if((!temp->cut)){
                                Set childcut value to true
                                break;
         }
}
```

record removemax()	
Description	Remove the max element in the tree and point the max to next maximum
Return Value	Return the pointer to the removed max

Pseudocode:

```
removemax(){
        temp = temp->right
        while(temp != maxp and maxp!==null)
                AddTotable(temp,ordertable) //add all elements to the table except maxp
        Temp = temp->child
        if(childs exits)
                AddTotable(temp.child,ordertable) // add first
        child while(temp!=max.child)
                AddTotable(temp->right,ordertable) // add all other childs
        Present = maxp;  //store max
        Maxp = null; // null max
        For (all elements in ordertable)
                insertfheap(ordertable.second) // u get new max becaus of insert
        Return present
        }
```

int main(int argc, char** argv)		
Description	Main function to handle input file and operations to be called according to input file and given a output	
Parameters	argc	Number of input parameters
	argv	Each paramerter
Return Value	0 or 1 if it is successfully finishes	

Pseudocode:

```
main(){
    N = Read each element from file
    if(N[0]==$)
        M = Read value from file
        if(N does not exists in hash table “hash”)
            Create new record temp with temp.value = M
            hash.insert(N,temp);
             insertfheap(temp);
        Else
            increment(hash.second,M)
    Else
        for(i=N to 1)
            P=removemax()
            Print P and store in a hash table
        “maxorder” for(all elements in maxorder)
            insertfheap(maxorder->second)
}
```

HashTagCounter

The HashTagCounter class contains the main function that runs the application.

The input is taken from a file and the operations on a heap are performed. This function writes output to a file after performing removeMax operation. This class is the encapsulating class for all other classes of the program. It contains the instances of all other classes to access their member functions.

CLASSES AND FUNCTION PROTOTYPES:

There are 2 classes that have been used to implement this project. They are:

- FiboHeapNode

- FiboHeap
- HashTagCounter

Here’s a short description of the functionality of each class.